a

**Lab Manual**

**Lab-III**

**Subject: Machine Learning**

**Subject Teacher: Dr. Abid Ali**

**Lab Supervisor: Miss. Sana Saleem**

**Linear Regression Model Implementation and Its Output Predictions:**

Objectives

* Implement linear regression to predict outcomes using a real-world dataset.
* Explore and preprocess datasets, including handling missing values and outliers.
* Apply data visualization techniques to understand data distributions and correlations.
* Train and evaluate a linear regression model using performance metrics.
* Use gradient descent to optimize model parameters and understand the cost function.
* Analyze model performance using metrics such as accuracy, precision, recall, and F1-score.
* Visualize the importance of features in the model and analyze training loss over time.

Pre-Requisite

* Read data from CSV files, clean missing values, and detect outliers using statistical techniques like Z-scores.
* Create visualizations such as histograms, correlation matrices, and scatter plots to explore dataset features.
* Implement linear regression using Python libraries and train-test splits, applying feature scaling techniques.
* Evaluate model performance using metrics like Mean Squared Error (MSE), R-squared, accuracy, precision, recall, and F1-score.
* Understand and implement gradient descent for optimizing linear regression parameters and visualize the cost function over epochs.
* Analyze feature importance using model coefficients and visualize their contribution to predictions.
* Develop and document a working machine learning solution for linear regression and understand the relevance of each step from data preprocessing to model evaluation.

Lab Practice 1: Implementation of Linear Regression Model with one variable

|  |
| --- |
| # Import necessary libraries  import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, r2\_score  from scipy import stats  # Load the Diabetes dataset  url = "diabetes[1].csv"  columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',  'DiabetesPedigreeFunction', 'Age', 'Outcome']  df = pd.read\_csv(url, names=columns)  # Display first few rows of the dataset  print(df.head())  # Basic statistics of the dataset  print(df.describe())  # Check for missing values  print(df.isnull().sum())  df = df.apply(pd.to\_numeric, errors='coerce')  df = df.dropna()  # Outlier analysis using Z-score for the selected feature 'Glucose'  z\_scores = np.abs(stats.zscore(df['Glucose']))  print("Outliers (Z > 3):")  print(np.where(z\_scores > 3))  # Remove outliers based on Z-score for 'Glucose'  df\_clean = df[(z\_scores < 3)]  print(f"Shape after removing outliers: {df\_clean.shape}")  # Splitting the dataset into the selected feature 'Glucose' and target variable 'Outcome'  X = df\_clean[['Glucose']] # Single input feature  y = df\_clean['Outcome'] # Output variable  # Train-test split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Train the Linear Regression model  model = LinearRegression()  model.fit(X\_train, y\_train)  # Predict on the test data  y\_pred = model.predict(X\_test)  # Calculate metrics  mse = mean\_squared\_error(y\_test, y\_pred)  r2 = r2\_score(y\_test, y\_pred)  print(f"Mean Squared Error: {mse}")  print(f"R-squared: {r2}")  # Visualizing model performance  plt.scatter(y\_test, y\_pred, color='purple')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2)  plt.xlabel("Actual")  plt.ylabel("Predicted")  plt.title("Linear Regression: Actual vs Predicted")  plt.show()  # Visualize residuals  residuals = y\_test - y\_pred  sns.histplot(residuals, bins=20, kde=True, color='orange')  plt.title('Residuals Distribution')  plt.show() |

Lab Practice 2: Implementation of Linear Regression Model with Linear Regression Practices.

|  |
| --- |
| import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, r2\_score  from scipy import stats  url = "diabetes.csv"  columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',  'DiabetesPedigreeFunction', 'Age', 'Outcome']  df = pd.read\_csv(url, names=columns)  print(df.head())  print(df.describe())  print(df.isnull().sum())  df = df.apply(pd.to\_numeric, errors='coerce')  print(df.isnull().sum())  df = df.dropna()  df = df.apply(pd.to\_numeric, errors='coerce')  df = df.dropna()  # Visualize the correlation matrix  plt.figure(figsize=(10,8))  sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)  plt.title('Correlation Matrix')  plt.show()  df.hist(bins=20, figsize=(14,10), color='blue')  plt.suptitle('Feature Distributions')  plt.show()  z\_scores = np.abs(stats.zscore(df))  print("Outliers (Z > 3):")  print(np.where(z\_scores > 3))  df\_clean = df[(z\_scores < 3).all(axis=1)]  print(f"Shape after removing outliers: {df\_clean.shape}")  df\_clean.hist(bins=20, figsize=(14,10), color='green')  plt.suptitle('Distributions After Removing Outliers')  plt.show()  # Splitting the dataset into features and target variable  X = df\_clean.drop(columns='Outcome')  y = df\_clean['Outcome']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Feature scaling (Standardization)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  model = LinearRegression()  model.fit(X\_train\_scaled, y\_train)  # Predict on the test data  y\_pred = model.predict(X\_test\_scaled)  mse = mean\_squared\_error(y\_test, y\_pred)  r2 = r2\_score(y\_test, y\_pred)  print(f"Mean Squared Error: {mse}")  print(f"R-squared: {r2}")  # Visualizing model performance  plt.scatter(y\_test, y\_pred, color='purple')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2)  plt.xlabel("Actual")  plt.ylabel("Predicted")  plt.title("Linear Regression: Actual vs Predicted")  plt.show()  # Visualize residuals  residuals = y\_test - y\_pred  sns.histplot(residuals, bins=20, kde=True, color='orange')  plt.title('Residuals Distribution')  plt.show()  # Training and Loss Visualization (Epochs Simulation)  epochs = 500  train\_errors = []  for i in range(epochs):  model.fit(X\_train\_scaled, y\_train)  y\_train\_pred = model.predict(X\_train\_scaled)  error = mean\_squared\_error(y\_train, y\_train\_pred)  train\_errors.append(error)  # Plot training loss  plt.plot(range(epochs), train\_errors, color='blue')  plt.xlabel('Epochs')  plt.ylabel('Training Error (MSE)')  plt.title('Training Error Over Epochs')  plt.show()  features = X.columns  importance = model.coef\_  plt.figure(figsize=(10,6))  plt.barh(features, importance, color='teal')  plt.title('Feature Importance in Linear Regression')  plt.xlabel('Coefficient Values')  plt.show() |

**Lab Practice 3: Checking for the Accuracy, Precision, Recall and F1 Score Values (We will discuss these in the next Lab)**

|  |
| --- |
| import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score, precision\_score, recall\_score, f1\_score  from scipy import stats  url = "diabetes.csv"  columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',             'DiabetesPedigreeFunction', 'Age', 'Outcome']  df = pd.read\_csv(url, names=columns)  df = df.apply(pd.to\_numeric, errors='coerce')  df = df.dropna()  z\_scores = np.abs(stats.zscore(df))  df\_clean = df[(z\_scores < 3).all(axis=1)]  X = df\_clean.drop(columns='Outcome')  y = df\_clean['Outcome']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  # Train the Linear Regression model  model = LinearRegression()  model.fit(X\_train\_scaled, y\_train)  y\_pred = model.predict(X\_test\_scaled)  y\_pred\_binary = [1 if pred >= 0.5 else 0 for pred in y\_pred]  accuracy = accuracy\_score(y\_test, y\_pred\_binary)  precision = precision\_score(y\_test, y\_pred\_binary)  recall = recall\_score(y\_test, y\_pred\_binary)  f1 = f1\_score(y\_test, y\_pred\_binary)  print(f"Accuracy: {accuracy}")  print(f"Precision: {precision}")  print(f"Recall: {recall}")  print(f"F1 Score: {f1}")  mse = mean\_squared\_error(y\_test, y\_pred)  r2 = r2\_score(y\_test, y\_pred)  print(f"Mean Squared Error: {mse}")  print(f"R-squared: {r2}")  plt.scatter(y\_test, y\_pred, color='purple')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2)  plt.xlabel("Actual")  plt.ylabel("Predicted")  plt.title("Linear Regression: Actual vs Predicted")  plt.show()  residuals = y\_test - y\_pred  sns.histplot(residuals, bins=20, kde=True, color='orange')  plt.title('Residuals Distribution')  plt.show() |

**Lab Practice: Add Cost and Gradient Descent Function in the Lab**

|  |
| --- |
| import pandas as pd  import numpy as np  import seaborn as sns  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  url = "diabetes[1].csv"  columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',             'DiabetesPedigreeFunction', 'Age', 'Outcome']  df = pd.read\_csv(url, names=columns)  print(df.head())  print(df.describe())  print(df.isnull().sum())  df = df.apply(pd.to\_numeric, errors='coerce')  df = df.dropna()  # Remove outliers using Z-score  from scipy import stats  z\_scores = np.abs(stats.zscore(df))  df\_clean = df[(z\_scores < 3).all(axis=1)]  print(f"Shape after removing outliers: {df\_clean.shape}")  # Splitting the dataset into features and target variable  X = df\_clean.drop(columns='Outcome')  y = df\_clean['Outcome']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test)  np.random.seed(42)  m, n = X\_train\_scaled.shape  theta = np.random.randn(n)  # Initial weights  bias = 0.0  # Initial bias  learning\_rate = 0.01  epochs = 1000  cost\_history = []  # Define the cost function (MSE)  def compute\_cost(X, y, theta, bias):      m = len(y)      predictions = np.dot(X, theta) + bias      cost = (1 / (2 \* m)) \* np.sum((predictions - y) \*\* 2)      return cost  # Define gradient descent function  def gradient\_descent(X, y, theta, bias, learning\_rate, epochs):      m = len(y)      cost\_history = []      for epoch in range(epochs):          # Make predictions          predictions = np.dot(X, theta) + bias            # Compute gradients          d\_theta = (1 / m) \* np.dot(X.T, (predictions - y))          d\_bias = (1 / m) \* np.sum(predictions - y)            # Update weights          theta -= learning\_rate \* d\_theta          bias -= learning\_rate \* d\_bias            # Calculate cost and save it for plotting          cost = compute\_cost(X, y, theta, bias)          cost\_history.append(cost)          if epoch % 100 == 0:              print(f"Epoch {epoch}: Cost = {cost}")        return theta, bias, cost\_history  theta, bias, cost\_history = gradient\_descent(X\_train\_scaled, y\_train, theta, bias, learning\_rate, epochs)  y\_pred\_train = np.dot(X\_train\_scaled, theta) + bias  y\_pred\_test = np.dot(X\_test\_scaled, theta) + bias  mse\_test = mean\_squared\_error(y\_test, y\_pred\_test)  print(f"Test MSE: {mse\_test}")  plt.plot(range(epochs), cost\_history, color='blue')  plt.xlabel('Epochs')  plt.ylabel('Cost (MSE)')  plt.title('Cost Over Epochs (Gradient Descent)')  plt.show()  # Visualize predictions vs actual values  plt.scatter(y\_test, y\_pred\_test, color='purple')  plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linewidth=2)  plt.xlabel("Actual")  plt.ylabel("Predicted")  plt.title("Gradient Descent: Actual vs Predicted")  plt.show()  features = X.columns  importance = theta  plt.figure(figsize=(10,6))  plt.barh(features, importance, color='teal')  plt.title('Feature Importance After Gradient Descent')  plt.xlabel('Coefficient Values')  plt.show() |

**Lab Tasks:**

Home Task need to be submitted through the Teams, add all screenshots and relevant results with explanation must be added from the system.

1. You are required to implement a linear regression model on an insurance dataset, following the procedures outlined in the lab manual. The goal is to analyze the dataset, implement the model, evaluate its performance, and visualize the results. You will need to document each step, providing insights and analysis based on the findings.
2. Dataset is attached with.

